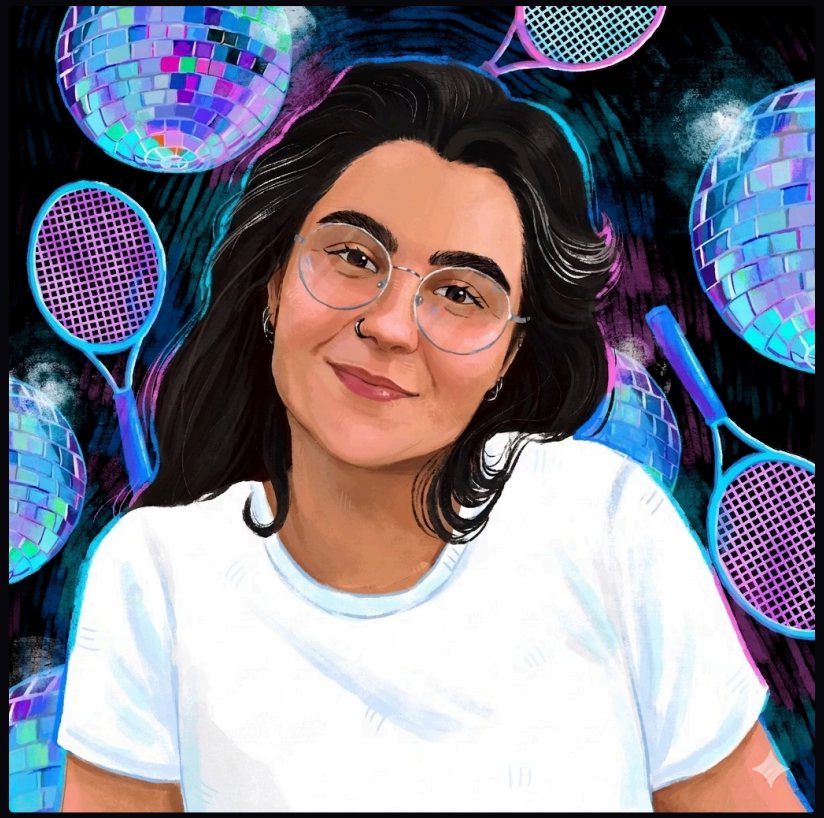
Cloud-to-Code

AWS Community Day Istanbul • May 9, 2026

INFRASTRUCTURE AS CODE

TERRAFORM • TERRAGRUNT



SPEAKER

Ceyda Düzgeç

Solution Architect | Platform Team @ Coca-Cola Içecek

🔗 Stalk me: **linkedin.com/in/ceydaduzgec**

Education

MS Software Eng. @ Boğaziçi •
BSc CS @ Sabancı

Certifications

AWS Solutions Architect Associate
HashiCorp Terraform Associate

Fun Fact

Today is my birthday! 🎂

Agenda

Problem: ClickOps Debt

Legacy infrastructure and the "Don't touch it, it works" culture

Farewell Terraformer

March 2026: Archived. Transition to new standards.

Path 1: Native Baseline

Terraform Import Blocks

Path 2: Visual Discovery

Former2: Free, visual

Path 3: SaaS Codification

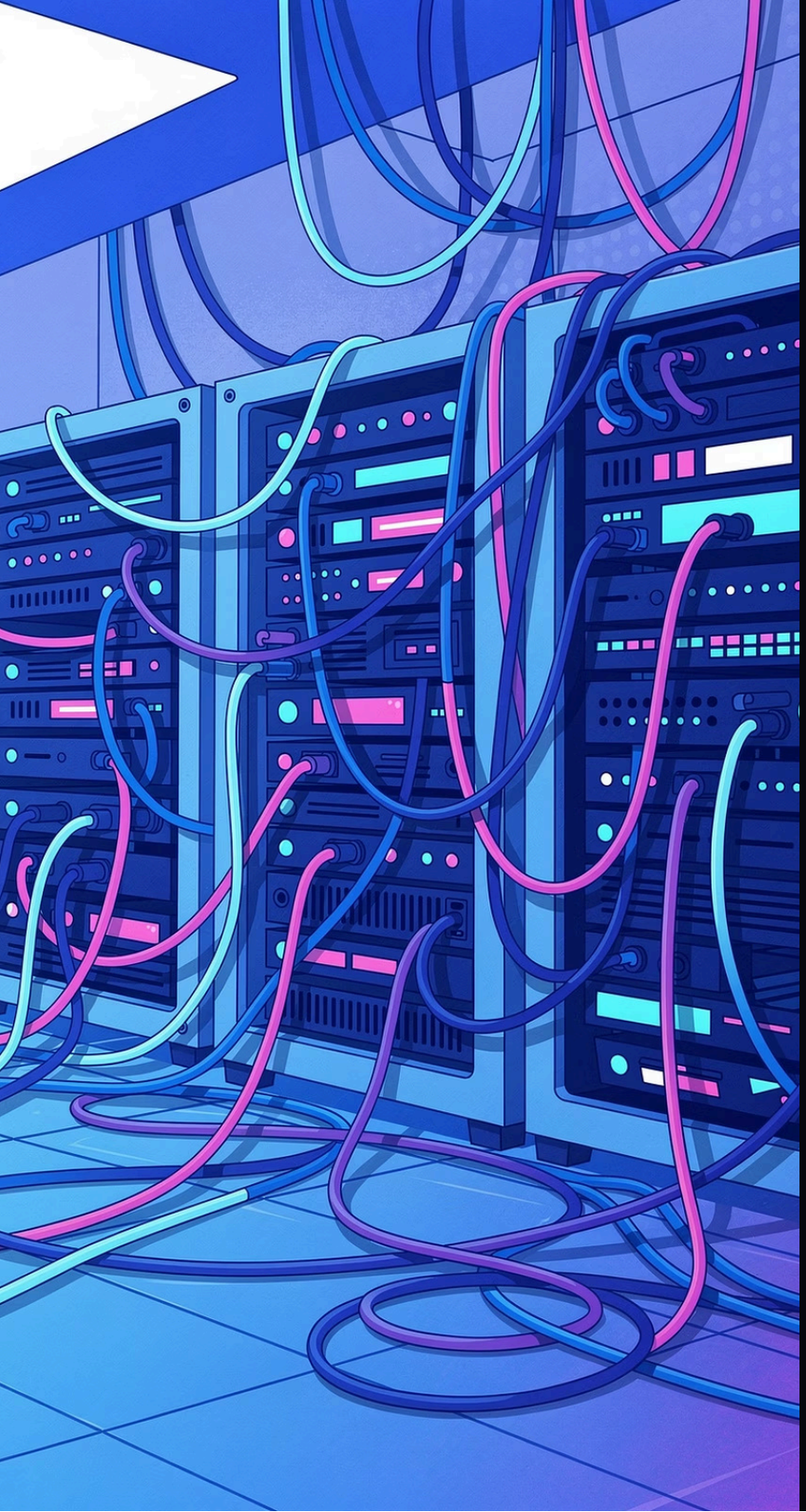
Firefly: Autonomous, modular, enterprise scale

Complementary Piece: Agentic IaC

Claude Code + Official HashiCorp Agent Skills

Production-Ready

Final touches with Terragrunt



Problem: Legacy by Console

> Legacy infrastructure is infrastructure built by clicking.

🚫 No .tf Files

The infrastructure exists only in the console, not in code

🌀 No State

Terraform is unaware of anything

👻 No Visibility

Who created what? Nobody knows

😱 'Don't Touch, It Works'

Fear of change kills engineering

Every resource created in the console is technical debt.

THE OLD WAY

Goodbye Terraformer 👋

⚰️ RIP Terraformer

Archived: March 2026

Officially unmaintained, no up-to-date provider support

Monolithic Code

Dumps all resources into a single file, hard to clean up

State Management

Imports it, but the config stays empty

✅ New Standard: Codification

Not Just Import

The goal now is to generate code, clean it up, and make it modular

Tool Ecosystem

Former2, Firefly

AI Layer

Claude Code + HashiCorp Agent Skills

> The goal is not just to import state, it's to codify infrastructure properly.

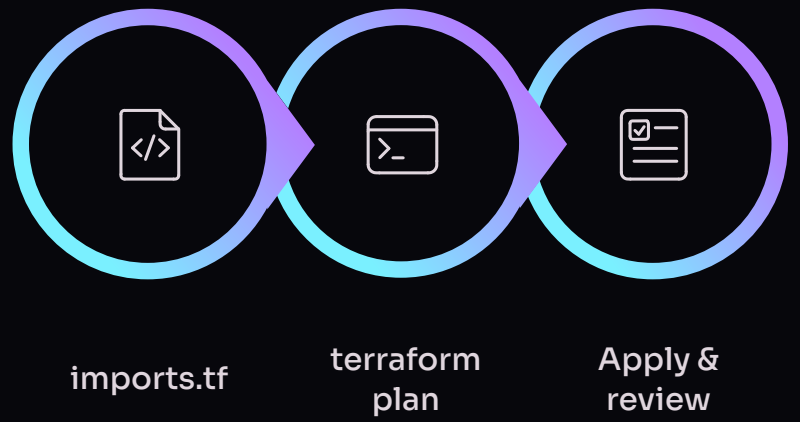
PATH 1

Terraform 1.5 – import block + generate-config

Best for: Known resource IDs, small number of resources

```
# imports.tf
import {
  to = aws_vpc.main
  id = "vpc-014738fc1473ba10f"
}
import {
  to = aws_instance.my_ec2
  id = "i-045d09d9cb1d65234"
}
```

```
terraform plan -generate-config-out="generated.tf"
terraform apply
```



PATH 1

Pros & Cons

✅ Pros

- Native Terraform → no extra tooling
- Import block lives in version control, gets code reviewed
- Visible at plan time → no surprises

⚠️ Cons

- Must know resource IDs upfront
- generate-config is **experimental** → code doesn't work out of the box
- Manual cleanup required every time

⚠ Config generation is experimental, always review output before applying.

PATH 2

Former2 - Visual Discovery

Best for: You don’ t know what exists • You prefer a visual workflow

1

Open former2.com

Runs entirely in your browser

2

Provide AWS Credentials

Nothing leaves your machine - Create credentials with ReadOnly access

3

Click "Scan"

Entire region is discovered automatically

4

Select Resources

Cherry-pick exactly what you need

5

Click "Generate"

Clean, working HCL is ready

PATH 2

Pros & Cons

✅ Pros

- Visual → see everything at a glance
- No need to know resource IDs
- Generates clean, working HCL
- Free and open source
- Generates multiple IaC languages

⚠️ Cons

- Browser-based → hard to automate
- Slow scanning on large accounts
- Security concerns with credentials

ⓘ Great entry point for teams new to IaC migration low friction, high visibility.

PATH 3

Firefly & AI-Powered Codification

Best for: Large, unknown infrastructure • SaaS-powered • Clean HCL at scale

```
# 1. Connect your AWS account
#    (read-only IAM role)

# 2. Firefly scans & maps all resources

# 3. Export clean Terraform HCL
#    with modules, naming conventions
#    and drift detection built-in

# 4. Open PR → review → merge
```

☁️ Full Account Scan

Discovers every resource across all regions automatically

🧹 Clean HCL Output

Generates module-ready code, not raw generated.tf

📡 Drift Detection

Continuously monitors for out-of-band changes

PATH 3

Pros & Cons

✅ Pros

- Full account scan → no manual ID entry
- Generates clean, module-ready HCL
- Built-in drift detection & alerting
- Integrates with GitHub PRs natively
- Supports AWS, GCP, Azure, K8s

⚠️ Cons

- SaaS product → not fully open source
- Requires read-only IAM role setup
- Free tier is limited; scaling requires a paid plan
- Overkill for small, known infrastructure

The Missing Piece

> "Former2, Firefly, generate-config — all generate code. None of it is production-ready on day one."

Tool Output	What's Still Missing
Raw .tf files	No module structure, hardcoded values
Imported state	Drift between state and reality
Generated HCL	No tflint, no fmt, no validate
Working code	Not idiomatic, not DRY, not reviewed

You still need something to review, fix, and validate the output automatically.

Solution: Claude Code + terraform-skill

Terraform

THE EXPERT LAYER

HashiCorp Official Agent Skills

In early 2026, HashiCorp released official Agent Skills for Claude Code, going far beyond a community skill.

The Origin

Anton Babenko's terraform-skill inspired HashiCorp to build official Agent Skills → now the standard for AI-assisted IaC.

Install in One Line

```
/plugin marketplace add hashicorp/agent-skills
```

What's New in 2026

Refactor modules, write Terraform Stacks, run tflint & validate, all from Claude Code.

What Does Agentic IaC Solve?

Conflicting Attributes

Automatically cleans up conflicts like tags vs tags_all

Hardcoded IDs

Moves fixed IDs into variables and locals

Naming Convention

Applies naming standards across the entire company

Zero-Refactor Output

Code generated by AI is directly ready for PR

> Anton Babenko's terraform-skill project inspired HashiCorp. Official Agent Skills were announced in February 2026.

DEMO

Review Generated Code with the Skill

You → "Find and fix the issues in this generated.tf"

Claude + terraform-skill responds with expert-level fixes automatically.

→ Catches `availability_zone_id` conflicts

→ Removes `tags_all` (computed attribute)

→ Suggests naming conventions

→ Restructures into Terragrunt-compatible modules

TERRAGRUNT

Moving to Terragrunt

Directory structure after migration:

```
infra/
├── terragrunt.hcl      ← root config
├── vpc/
│   ├── main.tf         ← cleaned code
│   └── terragrunt.hcl  ← inputs
└── ec2/
    ├── main.tf
    └── terragrunt.hcl
```

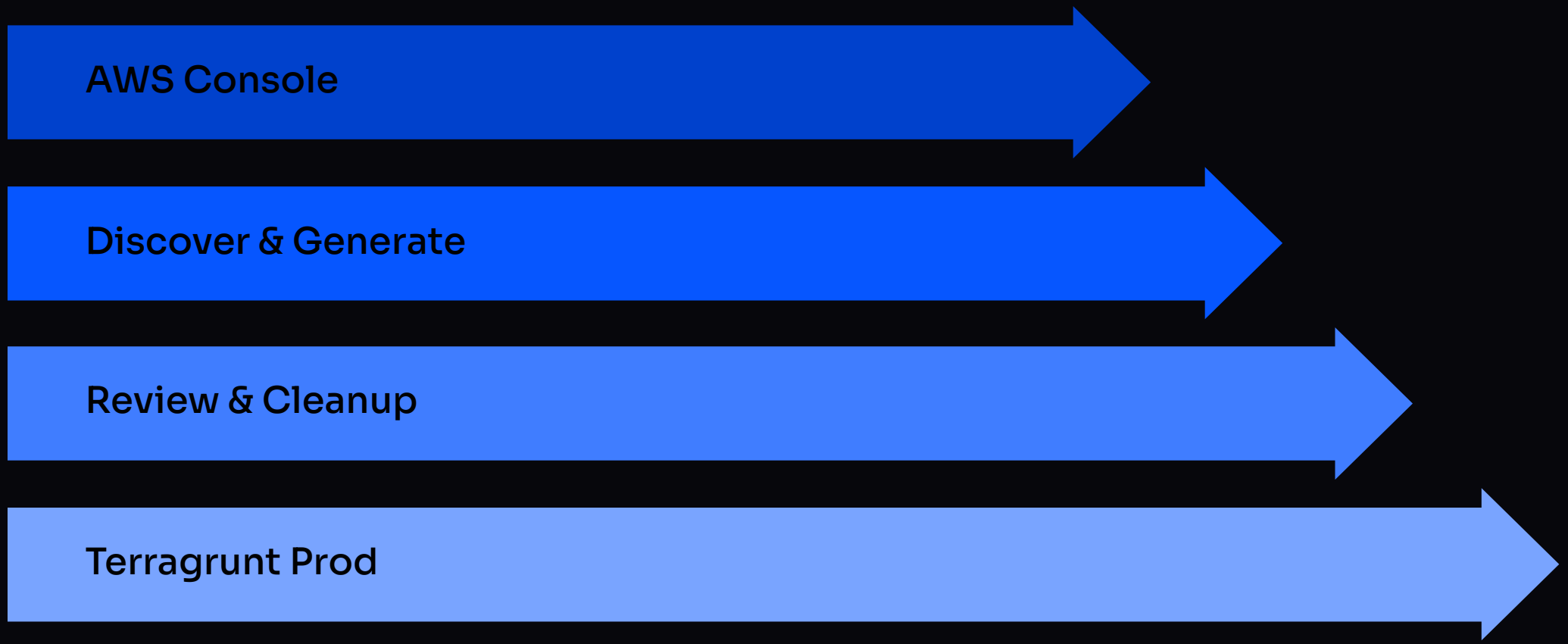
Import works the same way inside Terragrunt:

```
terragrunt plan \
 -generate-config-out="generated.tf"

terragrunt apply
```

Same import workflow, now with DRY configs, remote state, and dependency management.

The Loop Is Closed



Every step is intentional, auditable, and version-controlled from a Console click to a production Terragrunt module.

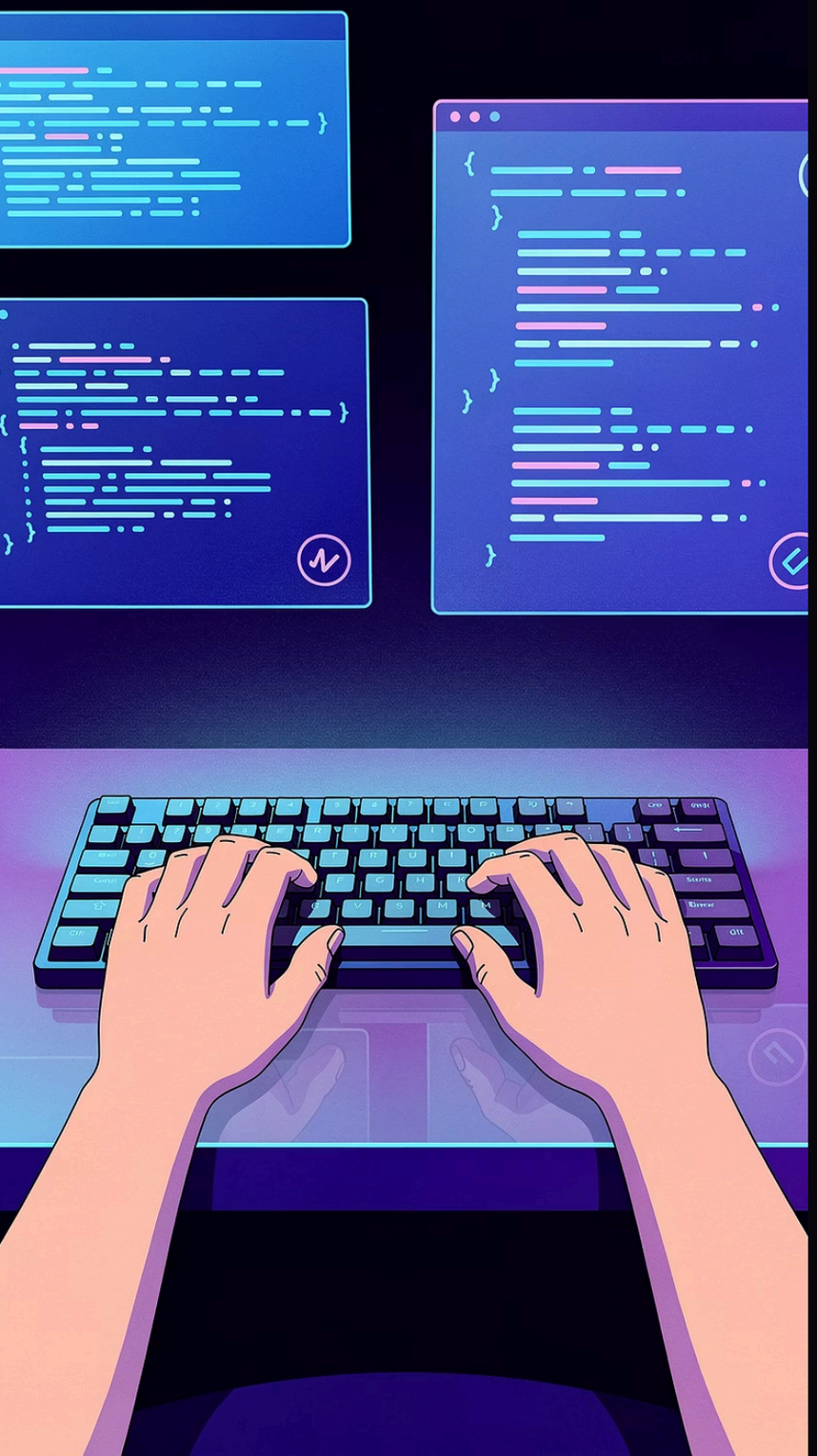
Summary: When to Use Which

Situation	Tool
IDs are known, 3-5 resources	Terraform Native
I need to explore (Visual)	Former2
Full Account / Enterprise	Firefly
AI review & auto-fix	Claude Code + Agent Skills
Multi-env, production-ready	Terragrunt

The common thread: **all three need a human review pass.**

None of them produce production-ready code on their own.

> Turning infrastructure into code is no longer a punishment — it's an AI-assisted process.



Try It Today

```
mkdir tf-import && cd tf-import

# Write imports.tf, then:
terraform init
terraform plan -generate-config-out="generated.tf"

# Install HashiCorp Agent Skills in Claude Code:
/plugin marketplace add hashicorp/agent-skills

# Then ask Claude:
# "Refactor this generated.tf into clean modules"
```

Start Small

Pick one known VPC or EC2 instance and import it today

Install Agent Skills

`/plugin marketplace add hashicorp/agent-skills` official HashiCorp plugin for Claude Code

Ship It

Wrap in Terragrunt and open a PR so your console clicks become code

Thank You!

Hope this helps you bring your cloud infrastructure under control 🚀

📁 Slides & Code

github.com/ceydaduzgec/presentations

🔗 Connect

linkedin.com/in/ceydaduzgec

❓ Questions?

Ask away!